Lec 21: Dynamic Programming – More Examples

Date: 23/01/23

Longest Increasing Subsequence (LIS)

Given a sequence $a_1, a_2, \cdots, a_n$, find longest increasing subsequence $a_{i_1}, a_{i_2}, \cdots, a_{i_k}$ such that $i_1 < i_2 < \cdots < i_k$ and $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$.

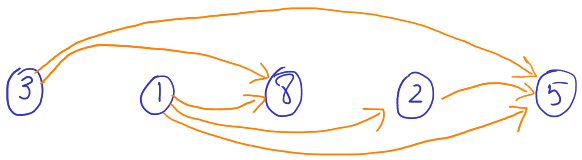
Examples: LIS([3 1 8 2 5]) ← 3 [1 2 5]

LIS([5 2 8 6 3 6 9 5]) ← 4 [2 3 6 9]

3 8
3 5
1 8
1 2 5 ← 3
2 5

DP Approach

Step 1: Visual Representation of the problem.



A directed acyclic graph (DAG) of numbers as nodes and edges from smaller numbers to larger numbers that appear later in the original sequence.

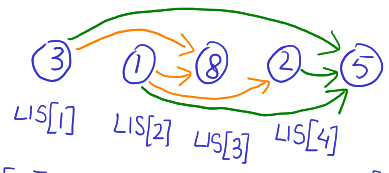
LIS = 1 + longest path in DAG

Step 2: Find appropriate subproblems

LIS[k] = LIS ending at index k

So,

LIS([3 1 8 2 5]) = LIS[5]



Using the DAG, to solve LIS[5], we will require the subproblems LIS[1], LIS[2], LIS[4] since node at index 5 can be legally added to any of those 3 subsequences to obtain a longer (by 1) increasing subsequence.

Step 3: Find relationships between the problem and subproblems.

This is the recursive breakdown!

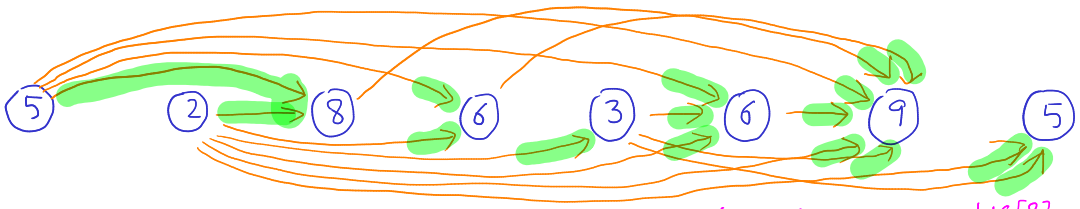
$$LIS[5] = 1 + \max(LIS[1], LIS[2], LIS[4])$$

Add 1 to the longest subseq. that can legally be extended by the node at index 5.

Step 4: Generalize the relationship

$$\underbrace{LIS[n]}_{\text{problem of size } n} = 1 + \max\{\underbrace{LIS[k]}_{\text{problems of size } < n} : k<n \text{ and } a_k < a_n\}$$

Step 5: Solve the subproblems in order until you get the answer to the main problem.



LIS[1] ← 1 LIS[2] ← 1

LIS[3] = 1 + max(LIS[1], LIS[2])
= 1 + max(1, 1)
= 1 + 1 = 2

LIS[4] = 1 + max(LIS[1], LIS[2])
= 1 + 1 = 2

LIS[5] = 1 + max(LIS[2])
= 1 + 1 = 2

LIS[6] = 1 + max(LIS[1], LIS[2], LIS[5])
= 1 + max(1, 1, 2)
= 1 + 2 = 3
bcz. of index 5

LIS[7] = 1 + max(LIS[1], ..., LIS[6])
= 1 + 3 = 4
bcz. of index 6

LIS[8] = 1 + max(LIS[2], LIS[5])
= 1 + 2 = 3

i_1	i_2	i_3	i_4
2	5	6	7

a_{i_1}	a_{i_2}	a_{i_3}	a_{i_4}
2	3	6	9

LIS →

1	2	3	4	5	6	7	8
1	1	2	2	2	3	4	3

Longest common Subsequence (LCS)

X = "ABBCFA"

Y = "ACBCCA"

LCS("ABBCFA", "ACBCCA")
= 1 + LCS("ABBCF", "ACBCC")
= 1 + max(LCS("ABBC", "ACBCC"), LCS("ABBCF", "ACBC"))

ABCA

LCS("A", "")

LCS("ABBCF", "")

LCS("A", "A")

LCS("ABB", "ACB") = 2

LCS("ABBCF", "AC")

LCS("", "") = 0

""	A	B	B	C	F	A
""	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0					
B	0					
C	0					
C	0					
A	0					?

Tabular Completion
↳ Final Answer via recursive thinking but iterative completion (bottom-up)

} Dynamic Programming